A Search Space "Cartography" for Guiding Graph Coloring Heuristics

Daniel Cosmin Porumbel, Jin-Kao Hao, Pascale Kuntz

A Search Space "Cartography" for Guiding Graph Coloring Heuristics

Daniel Cosmin Porumbel [1,2)], Jin Kao Hao [1)] and Pascale Kuntz [2)]

1) LERIA, Université d'Angers, 2 bd. Lavoisier, 49045 Angers, France

2) LINA, Polytech'Nantes, rue Christian Pauc, 44306 Nantes, France

August 7, 2009

Abstract

We present a search space analysis and its application in improving local search algorithms for the graph coloring problem. Using a classical distance measure between colorings, we introduce the following *clustering* hypothesis: the high quality solutions are *not* randomly scattered in the search space, but rather grouped in clusters within spheres of specific diameter. We first provide intuitive evidence for this hypothesis by presenting a projection of a large set of local minima in the 3D space. An experimental confirmation is also presented: we introduce two algorithms that exploit the hypothesis by guiding an underlying Tabu Search (TS) process. The first algorithm (TS-Div) uses a learning process to guide the basic TS process toward as-yet-unvisited *spheres*. The second algorithm (TS-Int) makes deep investigations within a bounded region by organizing it as a tree-like structure of connected spheres. We experimentally demonstrate that if such a region contains a global optimum, TS-Int does not fail in eventually finding it. This pair of algorithms significantly outperforms the underlying basic TS algorithm; it can even improve some of the best-known solutions ever reported in the literature (e.g. for $dsjc1000.9$).

Key words: graph coloring, local optima distribution, search by learning.

1 Introduction

The graph coloring problem (COL) is a well-known problem in Combinatorial Optimization. It is one of the 21 fundamental problems proved to be NP-Complete by Richard Karp in 1972 [31]. Given a graph G, COL asks to find the minimum number of colors (the chromatic number $\chi(G)$) necessary to color the vertices of G such that no two adjacent vertices share the same color. Due to its simplicity and generality, many practical applications have been modeled using coloring problems: timetabling [6, 13], scheduling [20, 33, 34], register allocation in compilers [7, 34], frequency assignment in cellular networks [24], air traffic flow management [2], supply chain management [35], etc.

The first algorithms for graph coloring were developed in the 1960s [5, 10, 41]. Since then, a considerable number of new methods have been developed and important progress

has been made. The second DIMACS Implementation Challenge [30] collected a large set of graphs that was used extensively since 1996 for benchmarking coloring algorithms.

Existing heuristic and metaheuristic coloring algorithms belong to three main solution approaches: sequential construction (algorithms like Dsatur [4], Iterated Greedy IG [12], RLX and XRLF [29]—very fast methods but not particularly efficient), local search (Tabu Search [3, 15–17, 28], Simulated Annealing [8, 29], Iterated Local Search [9, 38], Variable Neighborhood Search [1, 27, 40]) and evolutionary hybrid or population based search [14, 16, 17, 19, 36]. A recent survey, especially for the local search methods, is provided in [18].

We note that, very recently, two less-traditional local search algorithms proved surprisingly competitive [3, 27] and they matched many of the best results found so far in two decades. These local search algorithms focus on local level improvements—i.e. more powerful neighborhood relations [27] or alternative solution encodings [3]. Generally speaking, there are numerous local-level techniques that can enhance the performance of any local search heuristic.

However, local searches are also often substantially limited by the fact that they do not exploit enough *global* information about the search process. In most combinatorial optimization problems, local search methods are not typically very concerned about the relations between potential solutions (configurations) visited at *different* stages of the search. Moreover, most studies that analyze the search space structure (for example [11, 25, 26] for graph coloring) focus more on providing theoretical information than on effectively using such information to improve an algorithm.

Given any local search process searching through a search space, some important questions might be:

- what does the exploration path look like ?
- which regions is the process more likely to explore?
- can we be sure the search process does not explore only a limited number of regions?
- what is the spatial distribution of high quality solutions?
- are they randomly scattered or are they grouped in structures?

Our study consists in an exploration of these issues. The paper can be divided in two parts: the first one deals with search space analysis, and the second one effectively exploits the information from this analysis for the purpose of developing two new algorithms.

We analyze the spatial distribution of the high quality configurations (local optima) discovered by a classical Tabu Search (TS) algorithm for graph coloring (Tabucol) in a given period. The high quality configurations are compared with a distance measure—computed with a fast algorithm—and the resulting distance matrix is embedded in $\mathbb{R}^3$ (see Section 3.1) with a Multidimensional Scaling (MDS) procedure [32]. In the 3D representation, points represent colorings and the basic Euclidean distance between these points is an approximation of the computed distance between the associated colorings. The representations show that the high quality configurations are not randomly scattered in the space, but they form clusters that can be covered by spheres of specific diameter.

In the second part of the paper, we develop two algorithms that use a sphere-based search space organization. The first (TS-Div) is a classical Tabu Search (TS) algorithm

that keeps track of all the visited configurations by recording all visited spheres (Section 4). It also uses an additional learning process that guides the underlying Tabu Search process toward as–yet–unexplored spheres. TS-Int (Section 5) is a second-stage algorithm that performs in-depth explorations in the proximity of the high quality colorings provided by TS-Div.

The next section briefly outlines the formal details of the problem formulation, the local search algorithm (Tabu Search) and the test graphs. Section 3 presents the formal analysis of the spatial distribution of high quality colorings in the search space. Next, in Section 4 and Section 5, we present the two algorithms: TS-Div (assuring diversification) and TS-Int (assuring intensification). Numerical results and discussions are presented in Section 6, followed by conclusions in the last section.

2 Preliminary definitions and the general setting

2.1 Problem formulation and coloring representation

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The general graph coloring problem is to determine the chromatic number $\chi(G)$ of G, i.e. the minimum value of k for which there is k-coloring (i.e. a function $c : V \rightarrow \{1, 2, \cdots, k\}$) with no conflicts (i.e. with no $\{i, j\} \in E$ such that $c(i) = c(j)$).

In this study, we directly deal with the k-coloring problem: given a number of colors k, decide if G has a k-coloring with no conflicts. We start with a very large k (e.g. $k = |V|$) and we decrease k each time the corresponding k-coloring problem is solved. We recall the following definitions.

Definition 1 *(Array coloring representation) Given G and k, a coloring function $c : V \rightarrow \{1, 2, \cdots, k\}$ is encoded as an array $C = [c(1), c(2), \ldots, c(|V|)]$.*

Moreover, C is said to be a legal coloring if and only if $c(i) \neq c(j)$, $\forall \{i, j\} \in E$. A legal coloring represents a *solution* to the k-coloring problem. All colorings, both proper and improper, form the search space Ω of the problem.

Definition 2 *(Partition representation) A k-coloring $C = [c(1), c(2), \ldots, c(|V|)]$ of G is denoted as a partition $\{C^1, C^2, \ldots, C^k\}$ of V (i.e. a set of k disjoint subsets of V covering V) such that $\forall x, y \in V$, $x, y \in C^i \Leftrightarrow c(x) = c(y) = i$.*

We say that C^i is the class color i induced by the coloring C, i.e. the set of vertices having color i in C. This partition definition is particularly relevant to avoid symmetry issues arising from the color based encoding. It is consistent with the set-theoretic partition distance (see Section 4.2).

Definition 3 *(Objective function) Given a configuration C, we call conflict (or conflicting edge) any edge having both ends of the same color in C. The number of conflicts (or the conflict number of C) represent the objective (or fitness) function f for the k-coloring problem (G, k).*

A (G, k) problem is considered solved if and only if the algorithm finds a coloring such that $f(C) = 0$. A conflicting vertex is any $v \in V$, that has a neighbor of the same color.

2.2 Tabu Search for Graph Coloring

The search space analysis and the resolution methods from this paper are based on Tabucol [28], a classical Tabu Search for graph coloring [22]. Basically, Tabucol just passes from one configuration to another by modifying the color of a conflicting vertex, but it imposes the condition that each color change has to be not-Tabu, i.e. not performed in the near past. The general description of our TS version is presented in Algorithm 1; the stopping condition is to find a legal coloring or to reach a maximum number of iterations. The most important details that need to be filled are the neighborhood relation N and the Tabu list management.

Algorithm 1 Pseudocode General Tabu Search

Input: graph G, color number k, (optional) the start configuration C_{st}
Return value: $f(C^*)$ (i.e. 0 if a legal coloring is found)
C: the current coloring; C^*: the best coloring ever found
Begin

1. C:=C_{st} (choose C = *random configuration* if C_{st} is not specified.)
2. **while** a *stopping condition* is not met

   (a) find the best $C' \in N(C)$ so that move $C \stackrel{<i,i'>}{\longrightarrow} C'$ is not Tabu
   (b) $C = C'$ (i.e. perform move $< i, i' >$; $C(i) := i'$)
   (c) **if** $(f(C) < f(C^*))$
      - $C^* = C$
   (d) make the pair (i, i') Tabu for T_ℓ iterations

End

Neighborhood N Given a coloring problem ($G(V, E)$,k), the search space Ω consists of all possible colorings of G; thus $|\Omega| = |V|^k$. A simple neighborhood function $N : \Omega \rightarrow 2^\Omega - \{\emptyset\}$ can be defined as follows. For any configuration $C \in \Omega$, a neighbor C' is obtained by changing the color of a single *conflicting* vertex in C.

Definition 4 *(Plateau) We say subset $P \subset \Omega$ is a plateau of Ω if and only if: (i) all configurations in P have the same conflict number and (ii) for any $C_a, C_b \in P$, there exist configurations $C_1, C_2, \ldots, C_n \in P$ such that: $C_1 \in N(C_a)$, $C_2 \in N(C_1)$, ..., $C_n \in N(C_{n-1})$ and $C_b \in N(C_n)$.*

Tabu list management Tabu list is a structure used to record forbidden moves that have been performed in the recent past. For our problem, a move is characterized by a couple $< i, i' >$ meaning that the color $c(i)$ of a conflicting vertex i is changed to a new color i'. Each time a move $< i, i' >$ is realized, i is forbidden to receive again the color $c(i)$ for the next T_ℓ (Tabu tenure) iterations.

In our case, the Tabu tenure T_ℓ is dynamically adjusted by a function depending on the conflict number (f) (like in [14,15,17]), but also on the number m of the last consecutive moves that have not changed f. More precisely, $T_\ell = random(A) + \alpha * f(C) + \left\lfloor \frac{m}{m_{\max}} \right\rfloor$ where α and A are two parameters whose values are taken from [17] (i.e. $\alpha = 0.6$ and $A = 10$). The last term is introduced only to change T_ℓ when the algorithm is completely blocked looping on a plateau, or more exactly when f does not change for $m_{\max}$ moves. Each series of consecutive $m_{\max}$ ($m_{\max} = 1000$) moves with no conflict number variation triggers the increment of all subsequent values of T_ℓ (until f changes

again). This additional term prevents the search process from getting blocked looping on a plateau while not affecting its behavior outside plateaus.

2.3 The benchmark graphs

In this article, experimental studies are carried out on a variety of difficult problems from the well-established DIMACS Challenge Benchmark [30]: (i) $dsjcX.Y$—a series of classical random graphs [29] with unknown chromatic numbers (X denotes $|V|$ and Y denotes the density); (ii) $le450.25c$ and $le450.25d$—the most difficult "Leighton graphs" [33] with $\chi = 25$ (they have at least one clique of size χ); (iii) $flat300.28$ and $flat1000.76$—the most difficult "flat" graphs [12] with χ denoted by the last number (generated by partitioning the vertex set in χ classes, and by distributing the edges only between vertices of different classes); (iv) $R1000.1$—a geometric graph generated by picking random points (nodes) in the plane and by linking the points situated within a certain geometrical distance.

3 Cartography of the Graph Coloring Search Space

In this section, we explore the spatial distribution of the best configurations visited by TS in a given period of a single run. We consider these configurations as a set of points in the search space Ω (a $|V|$-dimensional space), and we measure the distance between each two points with the set-theoretic partition distance (see Section 4.2) that is a meaningful metric for graph coloring [17,21]. First, we provide 3D visualizations: the points from the $|V|$-dimensional space Ω are mapped into the 3D Euclidean space such that a distance distortion is minimized. This is achieved with a classical Multidimensional Scaling (MDS) procedure. Then, we analyze the values of distances between all points and we provide evidence that they are grouped in clusters in the search space Ω. We make an estimate of the cluster diameter. Surprisingly, we observed that it does not closely depend on the graph type, but mainly on $|V|$. Our assumption is that many of these clusters can be confined in spheres of radius $\frac{1}{10}|V|$.

3.1 Multidimensional Scaling

Multidimensional Scaling (MDS) is a common procedure in data visualization for representing similarities or dissimilarities in data. It takes as input a matrix of distances (or dissimilarities between pairs of items) and maps them to a set of locations in the Euclidean space ($\mathbb{R}^2$ or $\mathbb{R}^3$) such that a loss function (i.e. *Kruskall stress* in our case) is minimized. In our implementation, the MDS procedure has three steps: data collection, data mapping, and model verification.

Step 1: The matrix of distances in the real search space Given a sample of k-colorings $\{C_1, C_2, \ldots C_p\} \subset \Omega$, the procedure first constructs the matrix $D_{p\times p}$ where each element $D_{ij} = d(C_i, C_j)$ is the distance between C_i and C_j. We define the distance as the minimum number of color changes required to transform C_i in C_j. This distance was used by most other graph coloring studies (e.g. [17,21]); it is very suitable for the TS algorithm because it also represents the minimal number of moves TS needs to perform to arrive from C_i to C_j. It can be expressed as a generic set-theoretic partition distance

taking values between 0 and $|V|$; its computation can be done in $O(|V|)$ time—we refer to Section 4.2 for more details.

Step 2: Generating $\mathbb{R}^3$ coordinates To obtain the corresponding locations in $\mathbb{R}^3$, we use the classical *cmdscale* algorithm for metric (classical) multidimensional scaling as implemented in the well-known R programming language for statistical data analysis. The provided $\mathbb{R}^3$ locations are used to plot the 3D scatter graph and also to calculate the Euclidean distance matrix $d_{p\times p}$ between the points. The distances in the 3D representation assess the spatial distribution of the real configurations in the Ω space.

Step 3: Quality assessment Since the isometry between the Euclidean distance matrix ($d_{p\times p}$) and the initial distance matrix $D_{p\times p}$ can not be exactly satisfied, the quality of the embedding is measured with a goodness-of-fit indicator; we chose to use the classical stress [32]:

$$s = \sqrt{\frac{\sum_{1\leq i,j\leq p}(D_{ij} - d_{ij})^2}{\sum_{1\leq i,j\leq p} D_{ij}^2}}$$

The guideline provided by Kruskal in his seminal MDS paper [32] states that the representation is: *a*) poor if $s > 0.2$, *b*) fair if $s \leq 0.1$, *c*) good if $s \leq 0.05$, *d*) excellent if $s \leq 0.025$ and *e*) perfect if $s = 0$. In this paper, even if the total number of points is very high, we present no 'poor' (i.e. $s > 0.2$) representations.

3.2 Spatial distribution of configurations visited in limited-time runs

In this section we examine the sequence of colorings visited by (quite) short TS processes, more exactly we investigate the arrangement of high-quality configurations:

Definition 5 *(High-quality configuration) We say that configuration $C \in \Omega$ is a high-quality configuration (i.e. it is deep, or hard–to–find) if and only if $f(C) \leq B_f$, where B_f is a fitness boundary. Otherwise, we say that C is a low-quality configuration.*

Given a problem instance (G, k) *and* an initial high-quality coloring C_0, we apply the TS algorithm starting from C_0. TS visits a series of neighboring colorings and let C_0 ,C_1, C_2, ... denote the high-quality configurations, those satisfying $f(C_i) \leq f(C_0)$—i.e. we consider the fitness boundary $B_f = f(C_0)$. In all our tests, the number of high-quality configurations represents only a very small fraction of the total number of colorings visited along the search; we ignore the colorings worse than C_0 because they can be easily found.

We show in Figure 1 the MDS representations of the colorings resulting from this experiment. Two instances are considered: (a) random graph $dsjc1000.1$ starting from a 4-conflict coloring and (b) Leighton graph $le450.25c$ starting from a 1-conflict coloring. In order to limit the number of points (and the reliability of the MDS representation), we divide the TS exploration path in series of 100 and plot only the first coloring of each series. More exactly, if the search visits the following high-quality configurations in this order: $C_0, C_1, C_2, \ldots$, we graphically depict only $C_0, C_{100}, C_{200}, \ldots$.

These 3D representations provide a good intuitive image of the exploration path. In the left graph, the exploration process starts from the front-bottom-left corner and

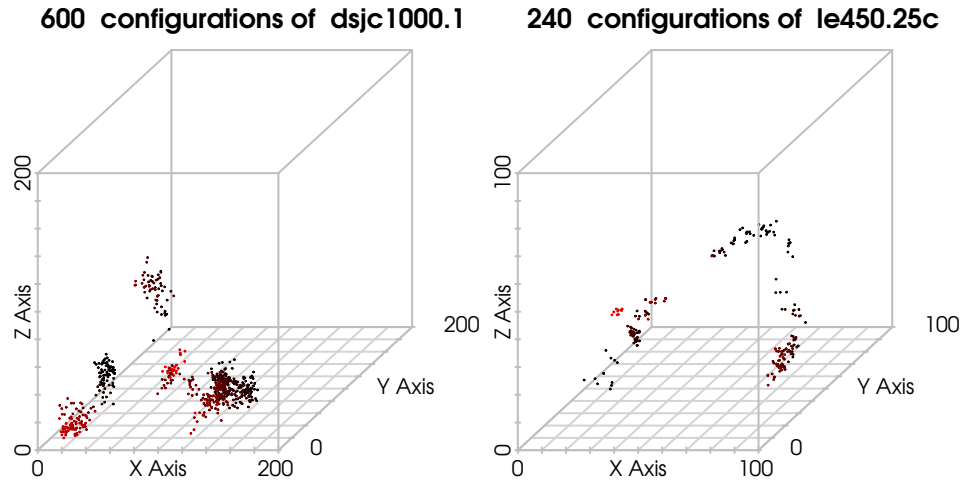

Figure 1: The high-quality colorings (with $f(C) \leq 4$) visited during 60000 iterations by TS for $G = dsjc1000.1$, $k = 20$ (left) and the high-quality colorings (with $f(C) \leq 1$) visited during 25000 iterations by TS for $G = le450.25.c$, $k = 25$ (right). The stress value is $s = 0.19$ and $s = 0.15$, respectively.

passes from cluster to cluster until it reaches the right side; most colorings visited in-between do not appear in the graph because they have worse fitness values (low quality configurations). In the right graph, the exploration path is even more clear (it starts from the front-bottom-left corner and ends toward the front-bottom-right corner) but the clusters are closer—for $le450.25c$, the distances between clusters are between $15\%|V| = 67$ and $22\%|V| = 99$ (see also Figure 2, bottom right graph) because the Leighton graphs have a peculiar structure as we discuss in Section 6.3.3.

3.3 Spatial distribution of colorings visited in long runs

Now, we present a more formal analysis of longer series of high-quality colorings visited by TS on *all* graph classes (Figure 2). We used a similar scenario as for Figure 1, but we examined much more configurations: indeed, *all* the high-quality colorings $C_0, C_1, \ldots, C_n$ (i.e. satisfying $f(C_i) < f(C_0)\ \forall i \in [1..n]$, where $n = 40.000$) visited by TS are now considered for analysis (instead of the several hundreds of samples $C_{100}, C_{200}, C_{200}, \ldots$ considered in the previous representations). We compute all distances[1] $d(C_i, C_j)$ and, with a distance histogram, we show how many pairs (C_i, C_j) correspond to each distance value.

Figure 2 shows bimodal distance distributions, with either very small or very long distances between the C_i's: this confirms the existence of some well separated regions with high densities of C_i's (clusters). If we denote a "cluster diameter" by c_d, we observe that c_d varies from $7\%|V|$ to $10\%|V|$ depending on the graph, such that: (i) there are numerous pairs (i, j) such that $d(C_i, C_j) < c_d$, (ii) there are very few (less than 1%) pairs (i, j) such that $c_d < d(C_i, C_j) < 2c_d$ and, (iii) there are numerous occurrences of some

[1]Considering $n = 40000$ colorings, the number of distances to compute is not extremely large, we computed all $n \times n = 1.600.000.000$ distances in several hours.

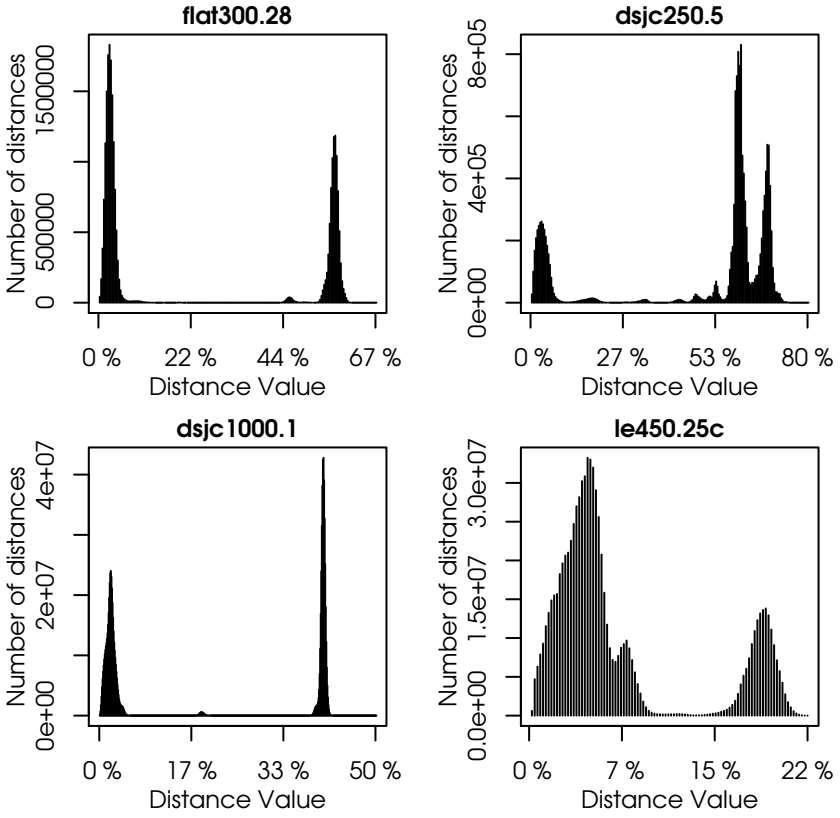

Figure 2: Histograms of the distances between each two of the $C_0, C_1, C_2, \ldots, C_{40000}$ (high-quality) configurations visited by TS; $f(C_i)$ is limited by: a) $f(C_0) = 4$ for ($G = flat300.28$, $k = 30$), b) $f(C_0) = 3$ for ($G = dsjc250.5$, $k = 28$), c) $f(C_0) = 4$ for ($G = dsjc1000.1$, $k = 20$) and d) $f(C_0) = 1$ for ($G = le450.25c$, $k = 25$).

larger distance values. This distribution of the C_i's mainly reflects the path of the search process through the search space—and not the arrangement of all existing high quality configurations. However, regarding the colorings visited by TS, it is important to note that any two C_i's distanced by more than $\frac{1}{10}|V|$ (the largest possible value of c_d) belong to *different clusters* of high-quality configurations. We keep this estimation in the rest of the paper and we use it to propose new search space exploration methods.

4 TS-Div—Tabu Search algorithm with diversification guarantee

While there are many well-studied methods to help a local search to escape a single basin of attraction, it seems more difficult to prevent it from *looping* between a limited number of basins of attraction. This is one of the main reasons for which, after a certain threshold, an increase of the execution time might not always improve the performance.

We present in this section a TS algorithm (call it TS-Div) that integrates a learning process for avoiding looping between already-visited areas. TS-Div employs an extended

Tabu list length if it detects that it comes upon configurations that are too close, according to the distance function, to other previously-visited configurations, i.e. while it passes through an already-explored cluster. This strategy prevents the algorithm from revisiting such regions by the reinforced diversification phase associated with the extended Tabu list.

4.1 Formal TS-Div description

The TS-Div algorithm (see Algorithm 2) is based on two central processes: (i) a basic TS exploring process (Tabucol), (ii) a reactive learning process that guides the first process by investigating the positions of the visited colorings. The orientation in space is achieved using the partition distance defined in Section 4.2 (basically, the distance between two colorings can be interpreted as the shortest path of TS steps between them).

Definition 6 *(Sphere) Given a (center) configuration $C \in \Omega$ and a radius $R \in \mathbb{N}$, the R-sphere $S(C)$ is the set of configurations $C' \in \Omega$ such that $d(C, C') \leq R$.*

Since we assume the clustering hypothesis (Section 3), we deal only with R-spheres of radius $R = \frac{1}{10}|V|$ in the rest of the paper. Two colorings C_a and C_b satisfying $d(C_a, C_b) \leq R = \frac{1}{10}|V|$ are called *close* or *related*; otherwise they are *R-distinct* or *R-distanced*. If $d(C_a, C_b) > \frac{|V|}{2}$, we say that C_a and C_b are *completely different*.

The first task of the learning component is to investigate the TS exploration path by recording the centers of all visited spheres. While configuration C, at the current iteration, stays in the sphere of the last recorded center C_p, we consider that the search process is "pivoting" around C_p. And while it remains in the same sphere of C_p, TS-Div explores the search space in the same way as the basic TS would do.

As soon as the search leaves the current sphere, the learning component concentrates on the guiding task. It first compares C to the archive of all previously recorded configurations (procedure `Already-Visited` in Algorithm 2) to check whether it is entering in a previously-explored sphere or not. If C is not in the sphere of any recorded configuration, it goes on only by changing the pivot—i.e. it replaces C_p with C and records it in the archive. Otherwise, if the search process is re-entering the sphere of a previously recorded configuration, the learning process intervene in the search process: a diversification phase is needed. For this purpose, we extend the Tabu tenure T_ℓ with a T_c factor as explained in the next section.

Diversification using the Tabu tenure Indeed, the Tabu list length (or Tabu tenure) provides a very simple mechanism to control diversification. Using longer Tabu lists makes configuration changes more diverse because TS never repeats the moves performed during the last $T_\ell + T_c$ iterations. Therefore, the greater the T_c value, the more diversification there is. A suitable control of T_c guarantees that TS-Div keeps discovering new regions; in fact, it can even guarantee that the process can never get stuck looping through already-visited spheres. The longer the time TS-Div spends only running into previously-visited spheres, the more it increments T_c—and T_c is only decremented (to 0) when the search process finds a new sphere. As such, the Tabu list increases continuously until a sufficiently high value of T_l is guaranteed to break any looping between already-visited

Algorithm 2 Pseudocode TS-Div

```
PROCEDURE ALREADY-VISITED
Input:  current configuration C
Return value:  TRUE or FALSE
  1. Forall recorded configurations C_rec:
       • If d(C, C_rec) ≤ R
           – Return TRUE
  2. Return FALSE
ALGORITHM POSITION-GUIDED TABU SEARCH
Input:  the search space Ω
Return value:  the best configuration C_best ever visited
C:  the current configuration
  1. Choose randomly an initial configuration C ∈ Ω
  2. C_p = C (the pivot, i.e. the last recorded configuration)
  3. T_c = 0  (the value by which it extends the Tabu tenure T_ℓ)
  4. While a stopping condition is not met
     (a) Set (next) C = the best non-Tabu neighbor in N(C)
     (b) If d(C, Cp) > R
           • C_p = C
           • If ALREADY-VISITED(C_p)
             – Then Increment T_c
           • Else
             – T_c = 0
             – Record C_p
     (c) Mark C as Tabu for T_ℓ + T_c iterations
     (d) If (f(C) < f(C_best))
           • C_best = C
     (e) If (f(C) < f(C_p))
           • Replace C_p with C in the archive
           • C_p = C (i.e. ''recentering'' the current sphere)
  5. return C_best
```

spheres (sooner or later). As such, TS-Div can discover new regions at all stages of the exploration, even in the very long run.

4.2 Distance definition and a fast computation method

The distance computation time is a crucial factor for TS-Div because, if this computation is too slow, it risks compromising the running speed of the algorithm. Fortunately, the set-theoretic partition distance fits well our purpose. Using the partition representation (see Definition 2), the distance between coloring C_a and C_b is the minimal number of vertices that need to be transferred from one class to another in C_a so that the resulting partition is equal to C_b (equivalent to the minimal number of moves needed by TS to arrive from C_a to C_b). There exists a well-studied distance computation method using an $O(|V|+k^3)$ Hungarian algorithm—see [23] for a general set-theoretic approach or [21] for the graph coloring application. However, in some conditions [39], the distance can be determined in $O(|V|)$ time with a special method.

Basically, the distance is calculated with the formula $d(C_a, C_b) = |V| - s(C_a, C_b)$, where s is a measure of similarity defined as follows. Using the definitions from Section 2, $s(C_a, C_b)$ is $\max_{\sigma\in\Pi} \sum_{1\leq i\leq k} M_{i,\sigma(i)}$, where Π is the set of all bijections from $\{1, 2, \ldots k\}$ to $\{1, 2, \ldots k\}$ and M is a matrix with elements $M_{ij} = |C_a^i \cap C_b^j|$ [21,23]. This similarity can be calculated by solving an assignment problem with the Hungarian algorithm. However, we applied the complete Hungarian algorithm only very rarely (less than 5% of all cases)

because the calculation can be simplified by taking into account some problem particularities. Matrix M has at most $|V|$ non-zero elements and, as also stated in [23, §2], they can be filled in $O(|V|)$ with the appropriate data structure. Indeed, the non-zero elements of M are situated at positions $M_{C_a(x),C_b(x)}$ (with $x \in V$) and our algorithm works only with these elements. Furthermore, we do not require here a precise distance value but we only have to decide if $d(C_a, C_b) > R$—i.e. $s(C_a, C_b) < |V| - R$. Since $s(C_a, C_b)$ is always less than $s'(C_a, C_b) = \sum_{1 \leq i \leq k} \max_j M_{ij}$ (because $M_{i,\sigma(i)} \leq \max_j M_{ij}, \forall \sigma \in \Pi$), in many cases it was enough to check $s'(C_a, C_b) < |V| - R$ to decide $s(C_a, C_b) < |V| - R$. All maximums $\max_j M_{ij}$ can be identified by going through the non-zero elements of M, and so, $s'(C_a, C_b)$ can be computed in $O(|V|)$. More conditions in which the problem can be solved in $O(|V|)$ are available in [39].

As such, a distance computation requires (in average) approximately the same computation time as a TS iteration. The next section describes a method to keep the number of iterations and the number of distance computations in the same order of magnitude during long TS-Div executions. Thus, our distance computation procedure guarantees that the slowdown introduced by the learning component stays in acceptable limits.

4.3 Running time of TS-Div

The most critical issue of TS-Div concerns its running speed, more exactly the slowdown introduced by the distance computations of the learning component. Our objective is to keep the time spent on distance computations in the same order of magnitude as the time spent on the exploring component. The most numerous distances are computed when TS-Div needs to check the distance from the current coloring to all colorings from the archive—see Algorithm 2, the `Already-Visited` procedure (`Forall` Loop in Step 1).

However, the archive processing time can be controlled if we focus the learning component only on the deep layers of search space, i.e. on the high quality configurations (with maximum B_f conflicts, see Definition 5). The fitness boundary B_f is automatically set by TS-Div in order to control the learning process overhead. More exactly, B_f directly controls the number of distance computations because, the whole learning component (Step 4.(b)) is now executed only if $f(C) < B_f$.

In all our experiments, TS-Div always sets B_f to a value so as to keep the number of distance computations in the same order of magnitude as the number of iterations. This proved to be a good "thumb rule" for obtaining an acceptable slowdown. In practice, we observed that B_f varies from 5 conflicts to 20 conflicts. The number of distance computations can be further reduced in many ways. For example, the distance computation in Step 4.(b) is not always needed: if $d(C_p, C) < R$, then TS needs at least $R - d(C_p, C)$ moves to get out of the sphere of C_p. As such, after each distance calculation in this step, TS-Div can safely skip it for the next $R - d(C_p, C)$ iterations.

5 TS-Int—A tree traversal of the search space

A typical issue of classical local search algorithms can be described as follows: the search process arrives in a local minimum situated in the sphere of a solution, and it needs to choose the next moves. In the fortunate case, the search process chooses moves on the

Algorithm 3 Pseudocode TS-Int

```
Input: the starting configuration C_s
Return value: the best configuration ever visited
Q = C_s
While Q is not empty
  1. C_s =FRONT(Q)
  2. Launch TS starting from C_s
       • Stop TS at the first high quality coloring C such that d(C,C_s) > 10%|V|
           – If d(C,C_i) > 10%|V| forall C_i ∈ Q
               * PUSH(C,Q) (insert it at the correct position as Q is sorted)
       • Stop TS if it gets too far from C_s
  3. If a stopping condition is met (i.e. enough processes launched from C_s)
       • POP(Q)
```

direction of the solution and, thus, the solution is soon discovered. But, there might exist millions of possible moves to escape from the local minimum; most of them might lead to different directions and the algorithm makes no distinction between them. Moreover, especially for TS-Div, if it chooses a move on a different direction and gets out of the sphere, it may never return again to it. To cope with this issue, TS–Int is introduced to methodically explore the proximity of a given starting coloring C_s.

First, TS-Int investigates the sphere $S(C_s)$ by launching from C_s a number of TS processes that explore only the proximity of C_s. The configurations discovered at the end of these processes constitute new starting points for the next exploration stages of TS-Int. All these new discovered starting points are kept ordered in a sorted queue Q (by their quality) and TS-Int processes them one by one (we say it performs a sphere exploration stage for each starting point). At each exploration stage, TS-Int takes the first configuration C_s in Q and launches a number of TS processes from C_s. Each such process is stopped if it finds a configuration C such that $C \notin S(C_s)$ and C is high quality (i.e. $f(C) \leq B_f$, where B_f takes the value typically set by TS-Div, see Definition 5).

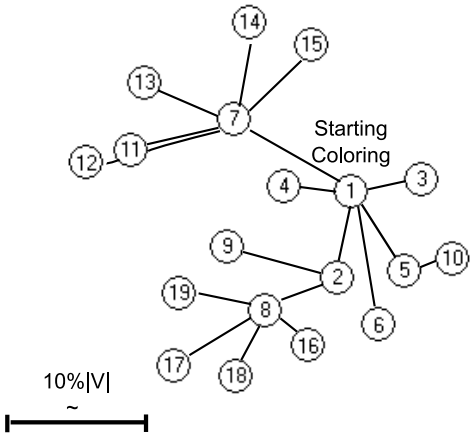

Figure 3: MDS representation (stress 0.05) of a TS-Int evolution for (le450.25,25). All points represent R-distanced 1-conflict colorings. They were discovered in the order of their numbers: 1 is the starting point, configurations $2, 3, \ldots, 7$ are discovered by the TS processes launched from point 1, etc.

When this condition is verified, we say that a new high quality configuration is discovered and, if there is no close configuration already in Q (i.e. if there is no $C_i \in Q$ such that $d(C, C_i) \leq 10\%|V|$), C is inserted into Q. We note that a TS process can also be stopped when it arrives too far from the center (i.e. if $d(C, C_s) > 20\%|V|$) without finding any high quality solution. After performing enough runs from one starting point, TS-Int takes the next configuration in Q and repeats (see Algorithm 3).

We can say that TS–Int organizes the search space in spheres that are processed in a methodical tree-traversal manner. C_s is the root of the tree and its edges link it to all new sphere centers discovered by TS processes started from C_s—recall that all sphere centers in Q are pairwise R-distanced. Similarly, each new discovered sphere center C_c constitutes a new tree vertex linked to the configuration from which C_c was reached. The degree of C_s corresponds to the number of pairwise R-distanced new sphere centers reached from C_s. Basically, TS-Int launches TS processes from C_s as long as there are chances to discover new configurations, R-distanced from all other configurations in Q.

Figure 3 illustrates a simplified TS-execution. Here, the solution, situated at distance $23\%|V|$ from the root vertex 1, is reached after an exploration of depth 3 (i.e. the tree on Figure 3 expanded on three levels). From experiments, the average degree of this exploration tree for this instance is actually 20. Consequently, the solution can be obtained by TS-Int after visiting at most $20^3 = 8000$ vertices. Indeed, we experimentally remarked that TS-Int always finds the solution in 10 runs out of 10 if it starts from a point within a distance of $\frac{1}{4}|V|$ from the solution (see more discussions in Section 6.3.2).

6 Results and Discussions

In this section, we present empirical results and discussions regarding both the search space structure and the two new algorithms. While TS-Div is a classical "stand-alone" algorithm, TS-Int is a second stage algorithm that can be used only to improve some existing configurations (typically provided by TS-Div).

6.1 Experimental procedures

First, we need to point out that TS-Div is equivalent to TS in the beginning of the exploration, while the archive is almost empty. The learning component intervenes in the exploration process only after a number of iterations, as soon as the exploration process stagnates. As such, if the basic TS can quickly find a solution, TS-Div does not find it more rapidly. The objective of TS-Div is visible in the long run, i.e. it only helps TS on the difficult instances where the basic TS fails.

To effectively test TS-Div, we perform 10 independent executions, each with a time limit of 50 hours. Within this time limit, TS-Div re-initializes its search with a random k-coloring each time it reaches 40 million iterations. Those restarts of TS-Div share the same archive of spheres. The statistics of the results are based on these 10 independent executions. Similarly, TS-Int is tested in the same experimental conditions, i.e. using 10 independent executions, each with a time limit of 50 hours. In Section 6.3.1, we will give some comments on the issue of running time with respect to the common practice of the literature.

Instance			TS-Div Algorithm		
Graph	χ, k^*	k	$\frac{successes}{executions}$	Iterations [$\times 10^6$]	Time [hours]
$dscj250.5$	?,28	28	10/10	4	< 1
$dsjc500.1$	?,12	12	10/10	42	< 1
$dsjc500.5$	?,48	48	2/10	7409	35
$dsjc500.9$	?,126	126	10/10	473	2
$dsjc1000.1$	?,20	20	2/10	2200	9
$dsjc1000.5$	?,83	87	5/10	2464	28
$dsjc1000.9$	?,224	224	8/10	1630	24
$flat300\_28\_0$	28,28	29	7/10	1186	8
$flat1000\_76\_0$	76,82	86	3/10	3020	33
$le450\_25c$	25,25	25	4/10	765	11
$le450\_25d$	25,25	25	2/10	1180	19
$r1000.1c$	?,98	98	10/10	47	< 1

Table 1: The results of TS-Div for a time limit of 50 hours. Columns 1, 2 and 3 denote the instance, the success rate (Column 4) is the number of successful executions out of 10; Column 5 and 6 show the average number of iterations and the average time needed to find a solution.

Regarding TS-Int, recall that it requires an input configuration C_s from which TS-Int starts searching for a solution (only a proximity of C_s is explored). As such, the successfulness of TS-Int depends entirely on the distance from C_s to the solution. Typically, if TS-Div fails on an instance, our procedure collects the best colorings ever visited and tries to improve them with TS-Int. We usually filter these colorings so as to keep only completely different colorings as starting points for TS-Int; we present the results on the best input colorings.

6.2 Numerical results

Table 1 reports the detailed results of TS-Div for several difficult instances (G, k)—especially those not easily solved by the basic TS. Columns 1–3 describe the instance, i.e. Column 1 is the graph, Column 2 shows χ (the chromatic number, or "?" if unknown) and k^* (the best known k); Column 3 denotes the k for which we apply our algorithm. Columns 4–6 present the results of the algorithm, i.e. the success rate (the number of executions that solve the problem in 50 hours or less) in Column 4, the average number of iterations and the average time required to find a solution in Column 5 and 6, respectively.

Table 2 presents the results of TS-Int on several starting configurations whose proximity is explored. Columns 1–3 have the same meaning as for Table 1, Column 4 shows the amplitude of the improvement (the number of conflicts of the start and the *end*—or best visited—configuration), Column 5 presents the success rate of achieving this improvement and Columns 6 and 7 denote the average computing effort (in iterations and in CPU hours, respectively).

TS-Int can find the solution with 100% success rate (see Column 5 of Table 2) when it starts from an appropriate coloring (i.e. not too far from a solution, see also Section Section 6.3.2). Remarkably, it finds a new legal coloring with 223 colors for the large graph $dsjc1000.9$. While TS-Div assures diversification, TS-Int is an intensification algorithm

Instance			Improvement	Success rate	Iterations [$\times 10^6$]	Time [hours]
G	χ, k^*	k	$f_{\text{start}} \longrightarrow f_{\text{end}}$	$\frac{successes}{runs}$		
$dsjc1000.1$	$?, 20$	20	$1 \rightarrow 0$	10/10	3774	12
$dsjc1000.5$	$?, 83$	86	$2 \rightarrow 0$	10/10	623	19
		85	$80^{*(k+1)} \rightarrow 0$	2/10	1453	39
$dsjc1000.9$	$?, 223$	223	$1 \rightarrow 0$	10/10	23	4
$le450.25c$	$25, 25$	25	$1 \rightarrow 0$	10/10	3410	10
$le450.25d$	$25, 25$	25	$1 \rightarrow 0$	10/10	6466	25
$flat300.28$	$28, 28$	28	$150^{*(k+2)} \rightarrow 0$	10/10	< 1	< 1
$flat1000.76$	$76, 83$	85	$74^{*(k+1)} \rightarrow 0$	10/10	1655	36

Table 2: Instances for which colorings are improved by TS-Int using a time limit of 50 hours. The input colorings are typically provided by TS-Div; however, the cells marked * indicate that TS-Int finds a legal k-coloring only starting from a legal $(k+1)$ or $(k+2)-$coloring.

that can be systematically executed after TS-Div in order to (try to) improve its best colorings.

6.3 Discussion

6.3.1 Running time of TS-Div and TS-Int

In our experimentations, TS-Div and TS-Int were allowed to run 50 hours per execution for a given coloring instance. We see that within this maximum time, TS-Div and TS-Int are able to reach very competitive results for the set of difficult graphs. Moreover, not all solutions required the maximum 50 hours computation time.

Let us mention that in the literature on graph coloring, it is a common practice to run a coloring algorithm several hours to several days to (try to) solve a hard coloring instance. For example, some of the most recent coloring algorithms [3, 27] use running times of 10 hours for the largest instances. Now if we compare the number of iterations, the values required by TS-Div are even more comparable with respect to other local search algorithms. Indeed, the maximum iterations of TS-Div are in the order of billions (between 10^9 and 8×10^9) while the best local searches in the literature report the same order of magnitude (e.g. 2×10^9 iterations [3, Table 6] or 3×10^9 [27, Table 5]).

Now let us insist on a more important point of TS-Div concerning the running time. One understands that, by its very nature, TS-Div will continually explore new regions if it is given more computation time. Consequently, TS-Div will be able to find new or better solutions with the additional computation resources.

Notice that this is a desirable characteristic which is not verified by many existing algorithms. Very often, running them beyond some time (or iteration) threshold will not lead to better results simply because either the algorithms are trapped in deep local optima or because they re-explore again and again the same regions. TS-Div provides a simple, yet effective solution to this delicate issue because it is forced to discover new spheres at all stages of its execution. The same comment applies to TS-Int for which more computation time means more intensified exploitation of more spheres. Consequently,

better results can naturally be expected.

6.3.2 TS-Int—finding the global optimum from an approximate location

TS-Int can be quite useful even in combination with other algorithms, especially when one could provide (by any means) an approximate location of the solution. In our main experimentation procedure, we consider that the configurations with lower conflict numbers have more chances to be close to a solution; thus, TS-Int always processes the spheres in the order of their conflict number. However, another possible "guess" of the solution location is obtained by considering that a legal k-coloring might be close to a legal $(k+1)$-coloring or even to a $(k+2)$-coloring.

This assumption worked perfectly well for the $flat300.28$ graph for which TS-Int finds a legal coloring with $\chi = 28$ colors starting from a legal coloring with $\chi + 2$ colors (the colors greater than χ are replaced with color 1). The graphs in this family are generated by adding edges only between the χ independent sets of an initial $\chi-$partition of V [12]. A large proportion of the 30 classes of the legal 30-coloring are very close to some of the initial 28 independent sets and, thus, TS-Int can easily reconstruct the rest of the coloring; the distance between the legal 30-coloring and the legal 28-coloring is only $7\%|V|$.

We experimentally observed that, if we provide a starting point within $\frac{1}{4}|V|$ distance from a solution, TS-Int finds the solution with a 100% success rate. These facts were observed on several graphs and for different initial colorings within $\frac{1}{4}|V|$ distance from a known solution. Searching a solution within $\frac{1}{4}|V| = 25\%|V|$ distance (around the start configuration) typically requires a complete exploration of a tree on 3 levels because each edge corresponds to a distance of $10\%|V|$ in the search space. The number of levels that can be processed in a certain time limit depends on the exploration speed and on the average tree degree. Since TS-Int is also highly amenable to parallelization, one could speed up the algorithm by an order of magnitude by launching all processes from C_s in parallel. The required $\frac{1}{4}|V|$ precision of the solution location could thus be extended to $40\%|V|$ and even $\frac{|V|}{2}$ if sufficient computation time is allowed.

6.3.3 Search space structure specificities for each graph family

We observed that, even if the number of legal k-colorings is always small (for a difficult problem), the number of 1-conflict colorings can greatly vary. In the random graph case, before discovering a legal coloring, TS usually visits between 3 and 20 1-conflict colorings. For the $flat300.28$ flat graph, TS-Div always directly descends to the legal 30-coloring from a coloring with 4 or 5 conflicts; as such, it can discover a solution even without visiting any 1-conflict coloring at all. At the other extreme, for the Leighton graphs we found more than 1 million 1-conflict colorings for several 0-conflict colorings (this is why it seems easier to solve this problem by first finding a 1-conflict coloring with TS-Div and by applying TS-Int on it).

The explanation of these landscape differences lays in the structure of the graph to be colored. The Leighton graphs have a built-in 25-vertex clique [33], and numerous 1-conflict colorings might share a common conflict on this clique while being very different outside it. They form very large plateaus in which it is difficult to find the coloring that correctly colors the 25 vertices of the clique. This is also confirmed by the search space

analysis: the distances between the clusters of 1-conflict colorings (sequentially visited by TS, see Figure 2, Section 3) are significantly smaller than for the other cases.

The flat graphs are constructed from a χ-partition of V such that a legal coloring has to assign a different color to each set of vertices from the initial partition. Any 1-conflict coloring has already identified much of this partition and the transition to the solution is trivial; the descent to the solution is always very steep. For the random graphs, the numbers of 1-conflict and 0-conflict configurations visited by TS-Div differs by an order of magnitude.

7 Conclusions

We have performed an empirical space search analysis that shows that the local minima of the coloring problem are grouped in clusters covered by spheres of $\frac{1}{10}|V|$ diameter. We have devised a fast method to record the spheres of local minima visited by a local search process. The TS-Div algorithm records its exploration path (only by recording the spheres) and uses an additional learning process to discourage it from returning to already-explored spheres. Moreover, the TS-Div algorithm does not introduce any auxiliary user-provided parameters.

Graph	χ, k^*	Div/Int	VSS [27] 2008	PCol [3] 2008	ACol [19] 2008	MOR [37] 1993	GH [17] 1999	MMT [36] 2008
$dsjc250.5$	$?, 28$	*28*	–	–	28	28	28	28
$dsjc500.1$	$?, 12$	*12*	12	12	12	12	–	12
$dsjc500.5$	$?, 48$	*48*	48	49	48	49	48	48
$dsjc500.9$	$?, 126$	*126*	127	126	126	126	–	127
$dsjc1000.1$	$?, 20$	*20*	20	20	20	21	20	20
$dsjc1000.5$	$?, 83$	85	87	88	84	88	83	83
$dsjc1000.9$	$?, 224$	***223***	224	225	224	226	224	225
$le450.25c$	$25, 25$	*25*	26	25	26	25	26	25
$le450.25d$	$25, 25$	*25*	26	25	26	25	26	25
$flat300.28$	$28, 32$	*28*	28	28	31	31	31	31
$flat1000.76$	$76, 82$	85	86	87	84	89	83	82
$r1000.1c$	$?, 98$	*98*	–	98	–	98	–	98

Table 3: Comparison between TS–Div/TS–Int and the best results of the state-of-the art algorithms. All the reported colorings are available on the Internet for further research use: www.info.univ-angers.fr/pub/porumbel/graphs/tsdivint/

The main objective of TS-Div is the global diversification of the search process: unlike the basic TS, TS-Div does not risk redundant explorations in the long run. As such, TS-Div is much more effective than the basic TS and it competes well even with the best algorithms from the literature. The search capacity of TS-Div is reinforced by TS-Int, which is an intensification-oriented algorithm used to better explore the proximity of the most promising configurations. It is able to systematically find a solution if the solution is situated within a certain distance from the starting configuration. TS-Int organizes the

search space as a tree of connected spheres and applies a classical tree traversal algorithm to methodically explore all spheres one by one.

To summarize, we devised a pair of algorithms that assures both the diversification and the intensification tasks by guiding underlying search processes. Table 3 presents a comparison between the best results obtained in this study and the best results from the literature. Four papers in this table are very recent (year 2008) as we took into account only the best algorithms; moreover, some of these cited papers present in fact more than one algorithm version—but Table 3 shows the best k reported by any version. The population-based heuristics (last four columns) are traditionally the most effective and, on some instances, they all find better results than all local searches. However, our method is also very effective comparing with all state-of-the-art algorithms and it even finds a new legal coloring for a very hard DIMACS instance ($dsjc1000.9$).

The principles behind TS-Int and TS-Div can be applied to any combinatorial optimization problem because they are both extensions of the Tabu Search algorithm. The necessary condition to achieve this is to find a search space distance measure that: (i) has low computation complexity (comparing to a TS iteration) and (ii) is in accordance with the proximity from the perspective of the TS process (i.e. the distance should be equivalent to the number of TS steps between configurations).

Acknowledgments: This work is partially supported by the CPER project "Pôle Informatique Régional" (2000-2006) and the Régional Project MILES (2007-2009). We thank the referees for their useful suggestions and comments.

References

[1] Avanthay C, Hertz A, Zufferey N. A variable neighborhood search for graph coloring. *Eur J Oper Res* 2003;151(2): 379–88.

[2] Barnier N, Brisset P. Graph coloring for air traffic flow management. *Ann Oper Res* 2004;130(1): 163–78.

[3] Blöchliger I, Zufferey N. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comp. Oper Res* 2008;35(3): 960–75.

[4] Brelaz D. New methods to color the vertices of a graph. *Commun ACM* 1979;22(4): 251–6.

[5] Brown J. Chromatic scheduling and the chromatic number problem. *Manage Sci* 1972;19(4): 456–63.

[6] Burke EK, Elliman DG, Weare RF. A university timetabling system based on graph colouring and constraint manipulation. *J Res Comput Educ* 1994;27(1): 1–18.

[7] Chaitin GJ. Register allocation & spilling via graph coloring. In *SIGPLAN '82: Proceedings of the 1982 SIGPLAN symposium on Compiler construction*. New York, NY, USA: ACM, 98–105.

[8] Chams M, Hertz A, de Werra D. Some experiments with simulated annealing for coloring graphs. *Eur J Oper Res* 1987;32(2): 260–66.

[9] Chiarandini M, Stützle T. An application of iterated local search to graph coloring. In: Johnson DS, Mehrotra A, Trick M (Eds.). *Computational Symposium on Graph Coloring and its Generalizations*. 112–25.

[10] Christofides N. An algorithm for the chromatic number of a graph. *Comput J* 1971; 14(1): 38–9.

[11] Culberson J, Gent I. Frozen development in graph coloring. *Theor Comput Sci* 2001; 265:227–64.

[12] Culberson J, Luo F. Exploring the k-colorable landscape with iterated greedy. In: Johnson D, Trick M (Eds.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge* [30], p. 345–84.

[13] de Werra D. An introduction to timetabling. *Eur J Oper Res* 1985;19(2): 151–62.

[14] Dorne R, Hao JK. A new genetic local search algorithm for graph coloring. In *PPSN 98*, volume 1498 of *LNCS*. Springer, 745–54.

[15] Dorne R, Hao JK. Tabu search for graph coloring, T-colorings and set T-colorings. In: Voss S, et al. (Eds.) *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, 1998; p. 77–92.

[16] Fleurent C, Ferland J. Genetic and hybrid algorithms for graph coloring. *Ann Oper Res* 1996;63(3): 437–61.

[17] Galinier P, Hao JK. Hybrid Evolutionary Algorithms for Graph Coloring. *J Combin Optim* 1999;3(4): 379–97.

[18] Galinier P, Hertz A. A survey of local search methods for graph coloring. *Comp. Oper Res* 2006;33(9): 2547–62.

[19] Galinier P, Hertz A, Zufferey N. An adaptive memory algorithm for the k-coloring problem. *Discrete Appl Math* 2008;156(2): 267–79.

[20] Gamache M, Hertz A, Ouellet J. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Comp. Oper Res* 2007;34(8): 2384–95.

[21] Glass C, Pruegel-Bennett A. A polynomially searchable exponential neighbourhood for graph colouring. *J Oper Res Soc* 2005;56(3): 324–30.

[22] Glover F, Laguna M. *Tabu Search*. Springer, 1997.

[23] Gusfield D. Partition-distance: A problem and class of perfect graphs arising in clustering. *Inform Process Lett* 2002;82(3): 159–64.

[24] Hale W. Frequency assignment: Theory and applications. *Proc IEEE* 1980;68(12): 1497–514.

[25] Hamiez J, Hao JK. An analysis of solution properties of the graph coloring problem. In: Resende M, et al., (Eds.) *Metaheuristics: computer decision-making*. Kluwer Academic Publishers, 2004. p. 325-46.

[26] Hertz A, Jaumard B, de Aragão M. Local optima topology for the k-coloring problem. *Discrete Appl Math* 1994;49(1-3): 257–80.

[27] Hertz A, Plumettaz A, Zufferey N. Variable space search for graph coloring. *Discrete Appl Math* 2008;156(13): 2551–60.

[28] Hertz A, Werra D. Using tabu search techniques for graph coloring. *Computing* 1987;39(4): 345–51.

[29] Johnson D, Aragon C, McGeoch L, Schevon C. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Oper Res* 1991;39(3): 378–406.

[30] Johnson D, Trick M. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[31] Karp R. Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (Eds.). Complexity of computer computations. New York: Plenum Press, 1972. p. 85-103.

[32] Kruskal J. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 1964;29(1): 1–27.

[33] Leighton F. A graph coloring algorithm for large scheduling problems. *J Res Natl Bur Stand* 1979;84(6): 489–503.

[34] Lewandowski G, Condon A. Experiments with parallel graph coloring heuristics and applications of graph coloring. In: Johnson D, Trick M (Eds.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge* [30], p. 309–34.

[35] Lim A, Wang F. Robust Graph Coloring for Uncertain Supply Chain Management. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences—Track 3* 2005;3:81b.

[36] Malaguti E, Monaci M, Toth P. A Metaheuristic Approach for the Vertex Coloring Problem. *INFORMS J Comput* 2008;20(2): 302-16.

[37] Morgenstern C. Distributed coloration neighborhood search. In: Johnson D, Trick M (Eds.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge* [30], p. 335–58.

[38] Paquete L, Stützle T. An experimental investigation of iterated local search for coloring graphs. *EvoWorkshops*, volume 2279 of *LNCS*. Springer, 121–30.

[39] Porumbel DC, Hao JK, Kuntz P. An improved algorithm for computing the partition distance. *Submitted Paper (available on request)* 2008.

[40] Trick MA, Yildiz H. A large neighborhood search heuristic for graph coloring. In *CPAIOR 2007*, volume 4510 of *LNCS*. Springer, 346–60.

[41] Welsh D, Powell M. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Comput J* 1967;10(1): 85–6.